[este libro se distribuye conjunta e inseparablemente con el número 63 de la revista PCMANÍA]

{GUÍA PRÁCTICA}

PARA PROGRAMAR APLICACIONES EN C++

MANUAL DE INSTRUCCIONES DEL COMPILADOR DJGPP V.2.1

pcmanía

PRESENTACIÓN

En PCmanía sabemos bien que, en muchas ocasiones, la voluntad de aprender y de crear es mucho más fuerte que el poder adquisitivo de nuestros lectores, y por eso volvemos a ofreceros una guía sobre la nueva versión del compilador de C, C++ y Objective C de GNU, una herramienta de dominio público utilizada por programadores profesionales. Hemos incluido el software en el CD-ROM para que todos vosotros (tengáis conexión a Internet o no) podáis empezar a trabajar ahora mismo. Esperamos que esta guía sea de vuestro agrado.

[ÍNDICE]

4

[INSTALACIÓN DEL COMPILADOR] 7

[OPCIONES BÁSICAS DE COMPILACIÓN Y LINKADO] 12

[PROBLEMAS AL CREAR EL EJECUTABLE] 22

[ENSAMBLADOR BAJO GNU] 26

[MISCELANEA DE TEMAS SOBRE DJGPP] 31

{prólogo}

Hace bastante tiempo tuve el placer de hablar sobre DJGPP en la ya olvidada sección "Torre de Babel". Entonces lo presenté como el mejor compilador gratuito para PC del planeta y sigo manteniendo esta opinión que algunos lectores considerarán probablemente como demasiado entusiasta. Quienes crean esto seguramente nunca habrán tenido la ocasión de estudiar las cualidades de este paquete. Consideremos algunas de ellas:

A) Se trata de un compilador de C y C++ para desarrollar aplicaciones de 32 bits en modo protegido para MS-DOS.

B) La velocidad de los ejecutables que produce DJGPP (una vez que se han empleado correctamente las opciones de optimización) está a la altura de prestigiosos compiladores comerciales como Borland y Watcom. De hecho en el faq de Eli Zaretskii (faq210b) se menciona a la versión DOS de «Quake» como uno de los programas que han sido compilados con DJGPP (nota: no tengo razones para dudar de esta afirmación pero no he podido confirmarla).

C) DJGPP incluye un extensor que permite hasta 4 Gigas de memorial flat (lineal) y 256 Megas de memoria virtual.

D) Con algunas excepciones aisladas (como las librerías de clases de C++), DJGPP es de libre uso, no cuesta un céntimo y puede ser utilizado tanto para crear aplicaciones freeware, shareware o comerciales.

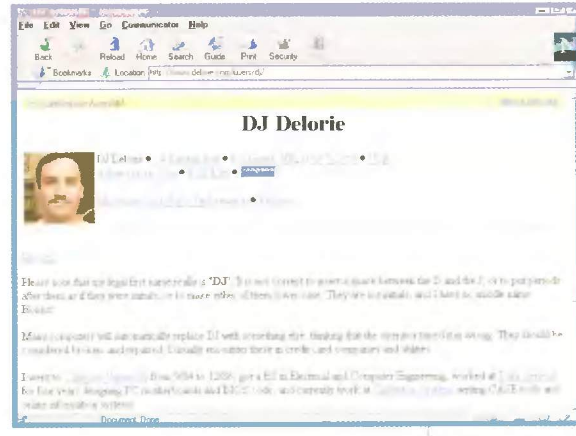
E) Los fuentes de DJGPP se incluyen con el mismo y cualquiera puede contribuir al desarrollo del compilador o a la caza de bugs (que siempre encontraréis en cualquier compilador sin que importe lo que este llegue a costar).

F) Además es igualmente posible conseguir depuradores, entornos de desarrollo integrados, utilidades diversas, librerías de todo tipo, etc. que, de igual forma, tampoco nos costarán un céntimo.

G) Dado que cualquiera puede contribuir al desarrollo de DJGPP, el crecimiento y mantenimiento de este producto, así como su adaptación a otros entornos, parece algo seguro.

La documentación de DJGPP fue diseñada y redactada por los mismos programadores que crearon el compilador, por lo que su estilo puede resultar algo difícil a los no iniciados

Estas características convierten a DJGPP en un producto único en su genero y en un compilador cuyo uso debe considerar seriamente cualquier programador que vaya a desarrollar aplicaciones bajo MS-DOS. En cuanto a sus puntos débiles, quizá el más molesto radique en su documentación. Esto no quiere decir que esta no exista sino que ha sido hecha por muchos programadores que obviamente disfrutaban mas programando que escribiendo doctoriales, los cuales pecan de una excesiva parquedad descriptiva en muchos puntos relativos al funcionamiento del compilador. Esto significa que no tendremos dificultades para averiguar el uso de tal o cual función, pero sí para saber por qué (por poner un ejemplo) DJGPP se cuelga en ciertos casos, o para averiguar determinadas cuestiones relativas al uso de las funciones DPMI o de la programación grafica. Este problema -que ya está desapareciendo gracias a la profileración de FAQs revisados por los gurús de DJGPP, a la aparición de páginas en internet sobre DJGPP y a la existencia de grupos de noticias (como comp.os.msdos.djgpp)- ha sido en parte la causa de ciertas confusiones en aspectos tan básicos como la procedencia e incluso el nombre de DJGPP. Y, ya que estoy hablando de confusiones, creo que lo mejor será que reconozca mi parte en ello puesto que en los primeros articulos sobre DJGPP se dejaron un poco en el aire estas cuestiones e incluso llamé erróneamente GNU a este compilador. Pido disculpas a todos por esta imprecisión de la cual soy el único responsable, y sólo puedo aducir en mi defensa es que en aquellos lejanos dias carecia de conexión propia a Internet y por tanto de la posibilidad de hallar información que pudiera sacarme de mi error. Incluso ahora, despues de haber leído un montón de textos sobre este particular y después de haber visitado bastantes "sites" sobre DJGPP, he encontrado muchas contradicciones e inexactitudes sobre este particular. Hasta hace poco, por ejemplo, yo pensaba que Delorie Software era un grupo de desarrolladores que habian escrito la mayor parte del compilador. Luego, al entrar en www.delorie.com, me asombré al descubrir que Dj Delorie era en realidad una pareja americana que habia decidido crear una versión para MS-DOS del GCC para UNIX de la FSF (más abajo comentaremos esto con algo más de detalle). Finalmente quiero aclarar que

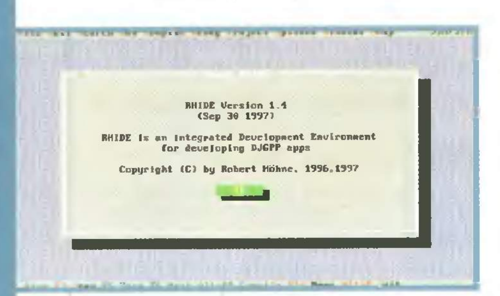


Actualmente la red Internet es una de las fuentes de información más completas en relación con GNU y todos los programas que se enmarcan en su licencia de software libre

me ha sido imposible comprobar personalmente todas las afirmaciones que vais a encontrar aquí (y que han sido tomadas de diversos FAQs, docs y paginas web). Ello hubiera requerido en rigor meses de pruebas pero, no obstante, tengo razones para creer en la certeza de dichas afirmaciones ya que proceden de gente que lleva mucho tiempo programando con DJGPP. Además todo lo que he leído hasta ahora en dichos archivos sobre la programación grafica, las optimizaciones, el uso de las funciones DPMI, etc., coincide con lo que pude comprobar por mí mismo durante las pruebas que realicé con la versión 1 que PCmanía publicó hace ya bastante tiempo. Aun así, por supuesto, acepto toda la responsabilidad por cualquier pifia que pudiérais (espero que no) hallar aquí.

ALGUNAS ACLARACIONES

DJGPP tiene copyright de Dj Delorie. Algunas partes de la librería libc.a tienen copyright de la Universidad de California. Otras utilidades y librerías adicionales creadas por terceras partes -y que no forman realmente parte de DJGPP- pueden ser halladas en los mismos directorios de los "sites" que han dejado espacio para este compilador y pueden llevar otros copyrights. GNU software (GCC, make, LIBG++, etc.) es copyright de la FSF (Free Software Foundation). No parece haber una distinción clara entre el GNU y DJGPP, y lo que parece desprenderse de los textos leídos por el autor de este artículo es que GNU es un proyecto de la FSF por el que este organismo intenta crear un conjunto de herramientas de software "libre" tan amplio como sea posible, y que DJGPP es una conversión del compilador GCC (perteneciente al paquete GNU) creado originalmente para UNIX (GCC es el compilador de C y C++). De hecho en el web de Delorie hay un archivo donde se explica muy someramente la historia de DJGPP y se cuenta cómo Delorie preguntó a la FSF si planeaban crear una versión del GCC para MS-DOS. La FSF respondió que GCC era demasiado grande para MS-DOS (estamos hablando de 1989) y ahí quedó la cosa hasta que Delorie creó la primera versión para un sistema operativo UNIX que corría en un 386. Luego Delorie escribio un extensor (go32) para el compilador y apareció la primera versión para MS-DOS que se llamó DJGCC y que solamente compilaba fuentes de C. Después, cuando el compilador se amplió para C++, este nombre fue cambiado por el de DJGPP (¡menuda sopa de nombres!). Pero todo esto ya es historia antigua y se han producido algunos cambios importantes entre la última versión publicada por PCmanía y la actual. Algunos de estos cambios afectan como ya veremos a la compilación de los fuentes antiguos de DJGPP en la nueva versión.



RHIDE es el entorno de trabajo que nos ofrece DJGPP para crear aplicaciones. Con el se facilita el trabajo de creacion y manipulacion de codigo fuente y algunas operaciones mas.

{capítulo 1} INSTALACIÓN DEL COMPILADOR

El sistema mínimo para instalar y utilizar DJGPP es un 386SX que tenga de 15 a 35 Megas libres de disco duro (o al menos eso dicen los FAQ). En la cruda realidad, sin embargo, pronto empezaremos a descomprimir archivos ZIP que se nos antojarán imprescindibles con lo cual la instalación requerirá mucho más espacio de disco. ¿Cuánto? Eso dependerá de cada uno.



A traves del propio entorno de trabajo podemos editar los documentos de ayuda e informacion que se incluyen en los archivos ZIP de distribucion

En cuanto a la RAM, según la documentación DJGPP puede funcionar con tan poca memoria como 4 Megas, aunque más nos vale no dar demasiado crédito a esta información. En el pasado alguna gente se quejó con razón de la lentitud de DJGPP en la compilación, siendo la falta de memoria el motivo de dicha lentitud. A estas alturas no creo que nadie tenga menos de 8 Megas en su PC, pero por si acaso daos por avisados.

En cuanto al entorno, el compilador correrá bajo MS-DOS o bajo cualquier sistema operativo que incluya DPMI. Algunos ejemplos citados son Windows 3.1, 95 y NT, OS2 y Linux (en su emulador de MS-DOS). A este respecto hay que añadir que bajo Windows, por ejemplo, las aplicaciones creadas con DJGPP correrán en ventanas de MS-DOS y que en dicho caso pueden presentarse ciertos problemillas muy específicos y relativos al modo en que Windows recupera la pantalla al "switchear" (perdóname este sacrilegio, Real Academia) entre la pantalla de Windows y la de MS-DOS. Otros problemas -debidos a errores en la configuración o al driver empleado- también pueden presentarse (y es en estos puntos donde la documentación resulta oscura, ¡snif!). Por otro lado resulta conveniente añadir que, según parece, ya resulta posible crear verdaderas aplicaciones Windows con DJGPP, aunque hay que utilizar herramientas auxiliares. Sin embargo este campo permanece completamente inexplorado por ahora para el autor de este artículo y por ello prescindiremos de dar más detalles sobre esto.

Pero volvamos al tema. ¿Cuál es la configuración óptima para usar DJGPP? Esto dependerá del equipo de cada usuario, pero el factor determinante parece ser la cantidad de memoria RAM instalada. En general se recomienda instalar los programas residentes y los drivers en la memoria alta y puede ser buena idea instalar un disco RAM si vamos a trabajar con fuentes de un tamaño considerable (aunque, de todas maneras, con las velocidades que tienen los discos duros de hoy en día esto puede resultar algo superfluo).

```
ile Edit Search Run Compile Debug Project Options Windows Help        91M/57M
]            e:/djgpp/contrib/rhide-1.4/copyrite.c                    2-[↑]
Copyright (C) 1995 DJ Delorie, see COPYING.DJ for details */
Copyright (C) 1996,1997 Robert Hohne, see COPYING.RH for details */
This file is part of RHIDE. */

d

lude <stdio.h>
lude <stdlib.h>
lude <string.h>
lude <sys/types.h>
lude <sys/stat.h>
lude <time.h>
lude <utime.h>
lude <limits.h>

fine CP \
 Copyright (C) 1996,%d Robert Hohne, see COPYING.RH for details %s\n" \
 This file is part of RHIDE. %s\n"

fine CO(start,end) sprintf(cline, CP, start, y, end, start, end)
 1:1
Save F3 Open F5 Zoom F6 Next Alt+F9 Compile F10 Menu Alt+X Quit
```

Una de las ventajas de trabajar con DJGPP es que se puede consultar el código de todo lo que se utiliza, incluso del propio RHIDE, uno de cuyos módulos se puede ver en esta imagen

POSIBLES INSTALACIONES

A estas alturas es de suponer que gran parte de los lectores tendrá ya conexión a Internet y por ello hemos querido respetar en nuestro CD la misma distribución de ficheros que podréis encontrar en ftp.simtel.net o en cualquiera de sus "mirrors". De esta manera aquellos que lo deseéis podréis, visitando periódicamente estas direcciones, mantener actualizada vuestra versión de DJGPP (podéis encontrar una larga lista de "mirrors" de SimTel en el FAQ de Eli Zaretskii que se guarda en el directorio \djgpp\faq con el nombre de djgppfaq.htm. También encontraréis una versión ascii de este FAQ con la extensión txt). Los directorios que usan el carácter 1 en el nombre se refieren a la versión anterior de DJGPP y los que emplean el 2 a la actual. El directorio v2 guarda la versión 2 de DJGPP y un par de extensores. En v2gnu hallaremos programas y herramientas creados por o para los compiladores GNU (recordad que GNU es un proyecto de desarrollo de software libre; GCC es el compilador de C y C++ pero aquí hay también otros compiladores como los de Fortran y Pascal). En el directorio v2tk encontraremos "toolkits" y librerías y en v2apps aplicaciones como el entorno RHIDE. Finalmente en v2misc encontraremos utilidades diversas. Dentro de todos estos subdirectorios situados a su vez dentro del directorio DJGPP hallaréis multitud de archivos ZIP y será escogiendo y descomprimiendo los archivos adecuados como instalaréis y personalizaréis vuestra versión de DJGPP. ¿Qué es lo que debemos instalar? Pues ello dependerá simplemente de lo que vayamos a utilizar. ¿Vais a programar en C o en C++? ¿Sois de los que utilizan depuradores? ¿Vais a compilar invocando manualmente a DJGPP o preferís usar un entorno integrado tipo IDE? ¿Qué librería gráfica preferís? Las opciones son muchas y crecen constantemente a medida que aparecen nuevas herramientas. Seguidamente indicaremos los archivos imprescindibles que habrá que descomprimir según algunos de los posibles casos entre los que podemos optar o que contienen archivos que vale la pena tener a mano (en cada caso se indica también el subdirectorio donde puede encontrarse el ZIP):

1) Para que un ordenador pueda ejecutar programas compilados por DJGPP necesitará el fichero v2misc/csdpmi3b.zip que contiene al Cwsdpmi, un servidor de DPMI de libre distribución creado expresamente para la versión 2 de DJGPP (realmente no necesitaremos a Cwsdpmi si ya tenemos otro servidor de DPMI pero se recomienda tener a mano este programa por si hay problemas). Otro fichero conveniente es v2misc/pmode11b que contiene otro servidor DPMI que podemos emplear en vez del anterior.

2) Conviene echar un vistazo al Djgppfaq de Eli Zaretski (en v2/faq210b.zip), el cual es quizá el FAQ más útil y completo que existe sobre DJGPP.

3) Para desarrollar programas en C (solamente) deberemos descomprimir los siguientes archivos: V2gnu/bnu27b.zip, que contiene una serie de utilidades como el ensamblador GNU (As) y el linker (Ld); v2/djdev201.zip, que contiene los archivos de cabecera de C, las librerías y utilidades diversas; v2gnu/gcc2721b.zip, que incluye el compilador de C e información sobre el mismo; y v2gnu/txi390b.zip, que contiene el programa Info, el cual nos resultará imprescindible para leer la documentación de GNU aprovechando el modo hipertexto en que está se halla dispuesta.

4) Para desarrollar programas C++ necesitaremos, además de los archivos anteriores, los dos siguientes: V2gnu/gpp2721b.zip, que contiene el compilador GNU C++, y v2gnu/lgp271b.zip, con los archivos de cabecera para C++, las librerías de clases y sus docs.

5) Los archivos djdev incluyen dos depuradores muy simples: el edebug y el fsdb, teniendo este último una interfaz parecida al de Turbo Debugger. Pero no nos vendrá mal cargar también con el GDB, un debugger de GNU que hallaremos en el archivo v2gnu/gdb416b.zip, junto a la información necesaria para manejarlo (esta herramienta puede ser empleada no sólo para depurar programas de C y C++ sino también para los creados con Pascal, Fortran y otros compiladores de GNU).

6) Una ayuda a la hora de compilar la tendremos en la herramienta Make, que hallaremos dentro del archivo v2gnu/mak375b.zip, junto con sus documentos.

7) En v2apps/rhide10b.zip encontraremos RHIDE, un entorno integrado de desarrollo muy similar (por no decir idéntico) al IDE de Borland. Esta herramienta escrita por Robert Hoehne incluye un debugger integrado (¡aleluya!).

8) Otro paquete que puede ser considerado como un entorno integrado es Emacs, el cual se describe en algunos sitios como el más poderoso y extensible editor que pueda emplearse en la programación (Nota: personalmente al autor de estas líneas le gusta más RHIDE). Este paquete está comprimido en los archivos de la serie v2gnu/em1934 .zip (y también será conveniente leerse el archivo v2gnu/emacs.readme).

9) Aunque ocupen 10 Megas, siempre puede ser conveniente disponer de los archivos fuente de la librería de las funciones de C de DJGPP. Estos fuentes están en v2/djlsr201.zip y gracias a su inclusión podremos corregir errores de librería (según parece, de cuando en cuando,

DJGPP V.2.1

alguien notifica algún error de este tipo en los foros de noticias de DJGPP).

10 | Para desarrollar aplicaciones en modo de texto y aplicaciones GUI (interfaz gráfica de usuario). GNU incluye los archivos siguientes: v2tk/grx20.zip, que es la librería gráfica de DJGPP; v2tk/bcc2grx.zip que incluye una librería para convertir las funciones de estilo Borland en llamadas apropiadas para la librería GRX (Nota del autor: no he comprobado este punto) y v2tk/pdc22.zip, donde tenemos otra librería de uso publico.

11 | También conviene tomar nota de la existencia de una serie de archivos con los que obtendremos unas utilidades cuyos nombres resultarán familiares a aquellos usuarios que conozcan los entornos UNIX. Con v2gnu/fil313b.zip, por ejemplo, obtendremos utilidades como ls, rm, cp, mv, y otras. Otros ficheros útiles para esto son v2gnu/bsh1147b.zip (el shell GNU), v2gnu/dif271b.zip (las "diffutiles" de GNU), y otros como v2gnu/flx252b.zip, v2gnu/grep20b.zip, v2gnu/shl112b.zip y v2gnu/txt119b.zip (si no sois amantes de UNIX podéis olvidaros de todo esto).

12 | Por otro lado, en caso de que queramos aplicar parches a las fuentes de DJGPP, deberemos descomprimir el archivo v2gnu/pat21b.zip.

Hay cientos de archivos FAQ, compilaciones de mensajes y listas de correo gracias a los que se puede resolver practicamente cualquier duda sobre el compilador DJGPP

Por supuesto existen muchos más archivos que los comentados aquí pero esto bastará para dar una idea al usuario acerca de lo que puede esperar de DJGPP. En caso de que queráis explorar el contenido de otros archivos, recordad que el número incluido en el nombre de los ficheros alude al número de versión que contiene y que dentro del CD puede haber varias versiones de un mismo programa. Además suelen emplearse también las letras b, s y d para aludir a "binaries" (los ejecutables), "sources" (los fuentes) y "docs" (documentación).

Ahora continuemos con el asunto de la instalación. Una vez que hayamos decidido qué archivos vamos a descomprimir crearemos un directorio llamado DJGPP donde los volcaremos desde el CD. Aquí conviene tomar nota de lo siguiente: si ya teníamos una versión antigua de DJGPP será conveniente borrar al menos el contenido del subdirectorio BIN antes de proceder a la descompresión de los archivos (o mover el contenido de BIN a otro sitio, si queremos conservarlo).

Según se afirma en varios FAQ, muchos antiguos usuarios de DJGPP han tenido problemas al haberse mezclado los ejecutables de la nueva versión con los de la antigua. Estos problemas se deben a los cambios realizados en la nueva versión de DJGPP que, en algunos casos, harán necesarias algunas modificaciones en los archivos fuente.

El siguiente paso será descomprimir los archivos con la opción "d" para respetar la estructura de directorios que debe crearse. Por ejemplo escribiremos: **pkunzip -d gcc2721b**

Una vez que hayamos descomprimido los archivos, colocaremos la variable de entorno DJGPP para apuntar al fichero DJGPP.ENV situado en el directorio raiz del compilador. También deberemos referenciar el subdirectorio BIN en la línea de paths, con lo que habremos de escribir en el archivo AUTOEXEC.BAT dos líneas parecidas a estas:

SET DJGPP=C:\DJGPP\DJGPP.ENV

SET PATH=C:\DJGPP\BIN;...

Después invocaremos al programa go32-v2 sin darle argumentos con lo que este nos indicará cuánta memoria DPMI y espacio Swap puede emplear DJGPP en nuestro ordenador. Si el programa da una cifra inferior a 4 Megas de DPMI... llorad e id pensando en comprar más RAM (y probablemente en cambiar de ordenador).

LA PRIMERA PRUEBA

Echad un vistazo a las siguientes líneas:

#include "stdio.h"

main(void) {printf ("hola mundo, he sido compilado con DJGPP v. 2");}

Este microprograma llamado hola.c es parecido al que los programadores de C emplean desde el alba de los tiempos para comprobar que un compilador funciona. Para hacer la prueba teclearemos: **gcc -o hola.exe hola.c**

Con esta línea DJGPP compilará y linkará el programa de una sola pasada creando el ejecutable hola.exe. Entonces, para ejecutarlo bastará con teclear "hola" y pulsar Enter (Nota: esta última frase puede parecer un insulto a la inteligencia del lector pero los usuarios ya experimentados de DJGPP recordarán que en la versión anterior hubiera sido preciso teclear "go32 hola" para ejecutar el programa. Con la nueva versión esto ya no es necesario. De hecho parte del antiguo código del extensor go32 forma ahora parte de los ejecutables generados por Djgpp v.2).

Por cierto, es muy posible que después de realizar la prueba anteriormente citada os parezca que la longitud del ejecutable resultante es un tanto excesiva (vuestro seguro servidor obtuvo un hola.exe de 80 Kb). Esto se debe a que por defecto DJGPP incluye en el ejecutable la información necesaria (los símbolos) para que las utilidades de depuración puedan trabajar. Así, si en vez de la línea anterior probamos con:

Gcc -s -o hola.exe hola.c

...obtendremos un ejecutable de 36 Kb. Esto sigue siendo bastante largo pero hay que recordar que hola.exe contiene dentro de sí parte del antiguo código de modo protegido de go32.

capítulo 2

OPCIONES BÁSICAS DE COMPILACIÓN Y LINKADO

En este capítulo comentaremos únicamente los detalles más indispensables y las opciones más básicas para poder crear un ejecutable.

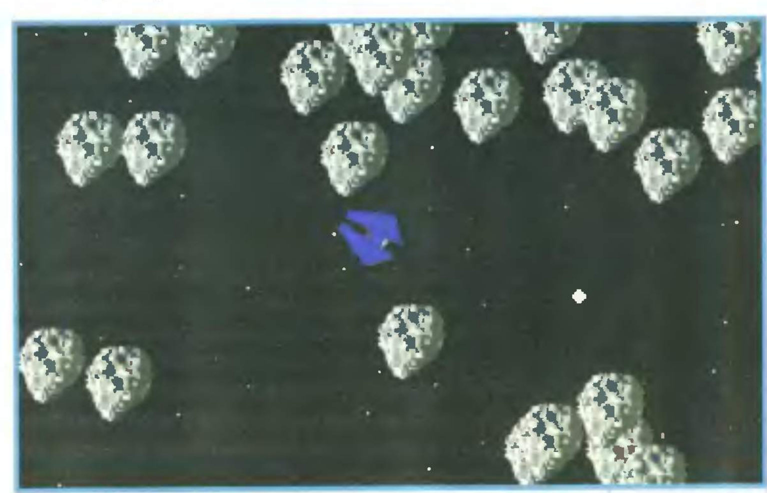
LA INFORMACIÓN

Si el lector ha descomprimido los archivos correspondientes, hallará la información acerca de todos los programas relacionados con DJGPP y sobre este mismo -así como la referencia de las funciones de C y C++ que admite DJGPP- en el subdirectorio INFO de su instalación de DJGPP. Esta información se encuentra en un formato de hipertexto por lo que necesitará usar el programa Info para poder acceder a ella. Si hemos descomprimido el archivo txi390b, este programa estará en el subdirectorio BIN de nuestra instalación de DJGPP y podremos invocarlo desde cualquier subdirectorio -tal y como sucederá con el compilador y las demás utilidades-, si nos hemos acordado de indicar en AUTOEXEC.BAT la ruta de acceso al susodicho directorio BIN.

El funcionamiento del programa Info es sencillo: al invocar al programa sin comandos aparecerá un texto por el que podremos movernos empleando las flechas. Entonces veremos en la parte inferior de la pantalla unas líneas que hacen referencia a una serie de temas relativos a DJGPP: la librería C estándar (libc.a), las utilidades (binutils), el compilador (GCC), etc. Como veréis, todos estos temas aparecerán precedidos por un carácter "*" para indicar que se trata de nodos. Los nodos son puntos de información relativos al texto situado a la derecha del asterisco. Dentro de un nodo cualquiera encontraremos a su vez otros nodos que subdividirán la información del nodo "padre" en subtemas. Así, partiendo del llamado "Directorio de nodos" (o nodo raíz), iremos moviéndonos de nodo en nodo -empleando los comandos de info- hasta encontrar la información deseada.

Veamos un ejemplo: supongamos que queremos informarnos sobre malloc(). Como estamos buscando información sobre una de las funciones de la librería emplearemos las teclas de flecha para situar el cursor sobre el tema libc.a. Entonces pulsaremos Enter, con lo que entraremos en el nodo de referencia de las librerías de C, y tomaremos nota de sus subnodos a fin de decidir a través de cuál de ellos podremos acceder a la información que estamos buscando. En este caso los subnodos son: Introduction, Functional Categories y Alphabetical List. Si decidimos probar con Functional Categories encontraremos un buen número de nuevos nodos en los que todas las funciones admitidas por DJGPP están divididas según su tipo: funciones matemáticas, de entrada-salida, de uso de ficheros, etc. Como malloc es una función relativa a la reserva de memoria, escogeremos el subnodo Memory Functions, dentro del cual hallaremos una extensa lista de nodos en la que cada nodo hará referencia a una función relativa a la categoría del nodo "padre". En dicha lista hallaremos un nodo con el nombre "malloc", y dentro del mismo tendremos la información buscada sobre dicha función, es decir, su prototipo, el fichero .h correspondiente cuya inclusión precisará la función, una descripción de la misma y un ejemplo práctico de utilización.

Sin embargo podemos encontrar una información cualquiera por más de un camino y algunos son más rápidos que otros. También, por ejemplo, podríamos haber buscado el nodo "malloc"



Algunas de las librerías disponibles para DJGPP están especializadas en la creación de juegos de acción como éste.

entrando en el nodo Alphabetical List antes mencionado. Una vez dentro podríamos desplazarnos con las teclas de avance de página y flechas hasta hallar "malloc" o bien podríamos teclear S, con lo cual ordenaríamos la búsqueda de una cadena, para teclear seguidamente "malloc", hecho lo cual Info nos situaría inmediatamente encima del nodo buscado. Otra solución aún más rápida sería pulsar "G" para impartir una orden de búsqueda del nodo deseado. Esto podremos hacerlo desde cualquier punto y únicamente será preciso recordar el fichero de información donde se halla el nodo. En este caso sabemos que malloc es una de las funciones de la librería, así que automáticamente sabremos que la información debe encontrarse en el fichero que se indicaba entre paréntesis en el nodo "libc.a". Por ello, después de pulsar "G", teclearemos "(libc.inf)malloc" y pulsaremos Enter, tras lo cual el programa Info nos situará inmediatamente en el nodo buscado.

Para entender esto situaos en el directorio de nodos (pulsad "d" si no estáis en él) y fijaos en los textos que aparecen entre paréntesis junto al título de cada nodo. Los textos se refieren a nombres de ficheros cuya extensión es .inf y que son los que emplea Info para funcionar.

Generalmente estos archivos tienen el mismo nombre que el nodo al que hacen referencia, aunque añadiendo la extensión inf. Así, por ejemplo, el fichero referente a las opciones de compilación de GCC es, por supuesto, GCC.INF. El problema que tiene el uso de "G" es que tendremos que recordar el nombre exacto del nodo que buscamos. Esto no supondrá ningún problema en el caso de las funciones pero sí en otros muchos en los que nos será más difícil recordar el nombre concreto. Hay que tener en cuenta, además, que la orden "G" precisará que indiquemos correctamente no sólo el nombre exacto del nodo sino también las letras mayúsculas y minúsculas dentro del mismo. Así, por ejemplo, si damos a "G" como entrada el texto "(libc.inf)Malloc" o (Libc.inf)MALLOC", Info nos dirá que no puede encontrar el nodo buscado. Por esta razón nos será muy sencillo buscar con "G" información sobre funciones de C (ya que sabremos casi con toda seguridad el nombre exacto a hallar) pero no nos resultará tan fácil emplear la misma opción con otros temas.

OPCIONES DE INFO	
H	Notas sobre Info.
?	Lista de opciones de Info.
N	Sirve para movernos al nodo siguiente al actual.
P	Sirve para movernos al nodo precedente al actual.
L	Nos desplazaremos al último nodo visto en la pantalla. (Nota: Esta opción no es demasiado exacta y probablemente no seguiremos el camino inverso al que recorrimos para llegar al nodo actual).
D	Nos situaremos en el directorio de nodos.
B	Ir al principio del nodo.
E	Ir al final del nodo.
G	Nos moveremos al nodo especificado por el texto dado como entrada. Al principio de dicho texto hay que especificar entre parentesis el nombre del archivo .inf donde suponemos que debe estar la información.
S	La búsqueda de cadenas. Daremos como entrada el texto de la cadena y luego pulsaremos enter para cada nueva busqueda.
Q	Salir de Info.

Otra opción útil para hallar rápidamente información sobre una función (sobre todo si no recordamos el nombre exacto) es "S". Después de pulsar la orden, Info buscará la palabra dada como entrada dentro del nodo actual. Normalmente, para que la búsqueda funcione deberemos estar ya dentro del fichero .inf donde esperamos que se halle la información. A diferencia de "G", "S" es indiferente al uso de mayúsculas o minúsculas.

El número de opciones que admite Info es sencillamente enorme (pulsad "?" para comprobarlo) pero normalmente nos bastará con saber emplear las opciones que podéis ver en la figura "opciones de Info".

¿Y en el caso de que prefiramos prescindir de Info? Bien, los archivos .inf son casi ASCII. Nos bastará, por tanto, con emplear nuestro editor ASCII favorito, aunque surgirán algunas dificultades: perderemos las facilidades del hipertexto y además nos será bastante difícil saber a priori en qué fichero se guarda una información dada. Para comprender esto editad el fichero GCC.INF con un editor ASCII. Como veréis lo único que este contiene son referencias de nodos que nos resultarán bastante inútiles mientras que la información buscada estará en uno cualquiera de los ficheros de la serie GCC.IX (de los que hay 26).

Otra posibilidad es emplear el entorno integrado de RHIDE. Para ello ejecutaremos RHIDE y pincharemos con el ratón sobre el menu Help. Tras esto aparecerá la ventana de ayuda desde la que podremos acceder a la información sobre las funciones (libc reference) o al índice de estas (Index for syntax help). También podremos solicitar ayuda acerca del funcionamiento del propio RHIDE (help) o sobre diversas informaciones como las opciones de optimización del compilador, la forma de crear un proyecto, la configuración de RHIDE, la herramienta Make, etc.

Por último hay que decir que Delorie mantiene una página con ayuda en línea acerca de DJGPP en http://www.delorie.com/gnu/docs.

C, C++ y OBJECTIVE C

GCC puede compilar fuentes de C, C++ y Objective C. Y de nuevo ello puede ser causa de confusión para el usuario que consulte la documentación, ya que a veces esta se refiere al compilador usando uno u otro término según los fuentes con los que haya de trabajar. Así, por ejemplo, cuando la documentación se refiere a la compilación de fuentes de C++, es usual que esta emplee el término G++ (G y no C) para referirse al compilador. En cambio el término GNU CC es utilizado cuando se quiere aludir al proceso de la compilación en general, sin señalar a particularidades debidas a un lenguaje concreto (Nota del autor: supongo que suelen emplearse estos términos porque MS-DOS no es el único soporte sobre el que corre el compilador. Las opciones de compilación y la documentación en general suelen aludir a cuestiones que afectan a todas las versiones del compilador y no sólo a DJGPP que, recordemos, es la versión de MS-DOS del compilador de la FSF. Ya sé que esto es lioso e incluso molesto, pero conviene que el usuario distinga bien estos términos o sufrirá confusiones al consultar la documentación).



Allegro es una popular libreria especialmente disenada para este compilador de C. Sus posibilidades para crear programas de entretenimiento son muy amplias.

Si nos aclaramos con esta maraña de términos no tendremos dificultades para entender la documentación, en concreto la parte que distingue la compilación de fuentes de uno u otro tipo. En cuanto al funcionamiento de GCC, por defecto incluye el linkado. La documentación describe la compilación de GCC como un trabajo en el que ocurren, por este orden, los siguentes procesos: preprocesamiento, compilación, ensamblado y linkado. Los tres primeros trabajan sobre los fuentes y el último genera un ejecutable a partir de los ficheros .obj que se han producido en los tres primeros pasos. GCC funciona por órdenes de línea y las opciones que indiquemos afectarán a los procesos antes mencionados. Decimos esto aquí porque precisamente pueden producirse problemas si el compilador se confunde al equivocarse con el tipo de lenguaje de los fuentes que debe compilar. ¿Cómo es esto posible? Pues porque GCC usa la extensión de los fiche-

DJGPP V.2.1

ros para determinar el tipo de lenguaje a tratar, admitiendo los siguientes tipos:

1) **cc, C, cxx y cpp son extensiones que indican fuentes de C++.**
2) **c es la extensión para fuentes de C.**
3) **m indica fuentes de Objetive-C.**
4) **S y s indican fuentes de ensamblador.**

Esto quiere decir que si, tenemos varios ficheros de C y empleamos un makefile para crear mediante Make el ejecutable, si usamos el carácter "C" como extensión en vez de "c", el compilador entenderá que estamos intentando compilar fuentes de C++ en vez de C y, con toda seguridad, se producirán errores. Pero, para evitar errores de este tipo, el compilador dispone de una opción con la que se puede especificar el lenguaje a tratar. Se trata de -x, a la que añadiremos alguno de los parámetros siguientes: c, objetive-c, c++, assembler y assembler-with-cpp.

MANEJANDO DJGPP

Como hemos dicho más arriba, GCC es un compilador "de línea" que efectúa por sí solo todos los procesos necesarios para crear un ejecutable. Dentro de los comandos de línea podemos especificar opciones que afectarán al preprocesamiento, a la compilación, al ensamblado o al linkado, y asimismo hay opciones que afectarán a los 4 procesos. En la mayoría de los casos estas órdenes de línea están pensadas para C pero también las hay para C++ y en ese caso esto se indica en la documentación (Nota: si habéis leído el apartado anterior sabréis, al examinar la documentación con Info, que las opciones de compilación para C++ están descritas en el nodo "Invoking G++"). Seguidamente comentaremos algunas (hay muchísimas) de estas opciones.

OPCIONES DE OPTIMIZACIÓN DE CÓDIGO

La manera más sencilla de controlar las optimizaciones es usar el comando de línea "O". Podemos dar a este comando un parámetro que especificará el grado de optimización que GCC debe aplicar. Podéis ver los diferentes casos en la figura siguiente, que muestra las opciones de optimización automática". Sin embargo es probable que el usuario prefiera controlar con mayor exactitud qué optimizaciones se activarán y cuáles no. Podéis ver algunas de estas opciones manuales en la figura de Optimización manual.

OPCIONES DE OPTIMIZACIÓN AUTOMÁTICA	
-O0	Las optimizaciones se desactiven.
-0 ó 01	El compilador intentará reducir el tamaño del código y su tiempo de ejecución.
-02	Esta opción optimizará el código aun más que la anterior. GCC activará todas las opciones manuales de optimización salvo las referentes a bucles y funciones en línea.
-03	Activa todas las optimizaciones posibles.

Nota del autor: hay programadores que pierden mucho tiempo jugando con opciones manuales de optimización y buscando la manera de conseguir el código más rapido. En mi opinión es bastante difícil saber a priori las consecuencias que va a tener el uso de una optimización concreta. A veces una opción de optimización -a causa de la propia forma del código- no nos servirá para ganar tiempo sino todo lo contrario. Y también puede ocurrir que la mezcla de varias optimizaciones sea contraproducente. Además hay algunos FAQ en los que se resumen los problemas más frecuentes de los usuarios y parece ser que algunos de ellos se deben a las optimizaciones. En efecto, según parece, a veces el uso de las optimizaciones puede ocasionar un cuelgue del compilador, especialmente si nuestro fuente es de C++ y la versión usada de GCC es la 2.6.0 (la publicada en el CD es la 2.7.2). Según algunos gurús de DJGPP, el problema puede estar en errores de sintaxis de nuestro código y que el compilador no localiza a tiempo. Por ello se recomienda usar #if y #endif para intentar localizar las zonas "peligrosas" de nuestros fuentes. Como caso curioso, añadiremos que estas afirmaciones han sido corroboradas nada más y nada menos que por el Povteam, el equipo de programadores autores del código del raytracer POV. En efecto: en un doctorial incluido en los fuentes, este equipo menciona que en las pruebas de compilación con la versión 2 de DJGPP, el ejecutable obtenido fue menos rápido que el resultante al compilar con la versión anterior de este compilador. Ello se debió a que -según afirman estos programadores- al emplear cualquier optimización el compilador se colgaba, con lo cual hubieron de desactivar a aquéllas quedando el ejecutable más lento con respecto al logrado con la versión 1.x, con la que no fue preciso desactivar las optimizaciones. Incluso así, el nuevo ejecutable daba problemas de vez en cuando (Nota del autor: no se especifica con qué versión de GCC intentaron compilar POV). Todo esto quiere decir que será mejor desactivar las optimizaciones mientras se esté trabajando en el desarrollo del programa y probar a activarlas al final, cuando estemos seguros de que todo lo demás funciona. Así pues ya sabéis: cuidado con las optimizaciones y estad atentos a los parches que próximamente pueda publicar la FSF (Nota: a pesar de estos problemas propios de la nueva versión, seguimos creyendo que esta representa un paso adelante con respecto a su predecesora, sobre todo porque -al igual que los miembros del Povteam- opinamos que los bugs de optimización no seguirán infestando mucho tiempo a este estupendo compilador).

OPCIONES DE OPTIMIZACIÓN MANUAL	
-fforce-mem	Fuerza a los operandos en memoria a que se copien en registros antes de que se efectúen operaciones aritméticas. La opción -o2 aautomática activa esta opción.
-fforce-addr	Fuerza a las constantes de direcciones de memoria a copiarse en registros antes de efectuar operaciones aritméticas.
-fno-inline	Hace que al compilador no preste atención a la palabra "inline" (por las funciones en línea).
-ffast-match	Activa todas las optimizaciones posibles.
-fstrengh-reduce	Realiza las optimizaciones de reducción de bucles y eliminación de variables de iteración.
-fno-function-cse	Evita al poner las direcciones de las funciones en registros haciendo que dichas direcciones, en las llamadas a las funciones, estén puestas explícitamente en las llamadas.

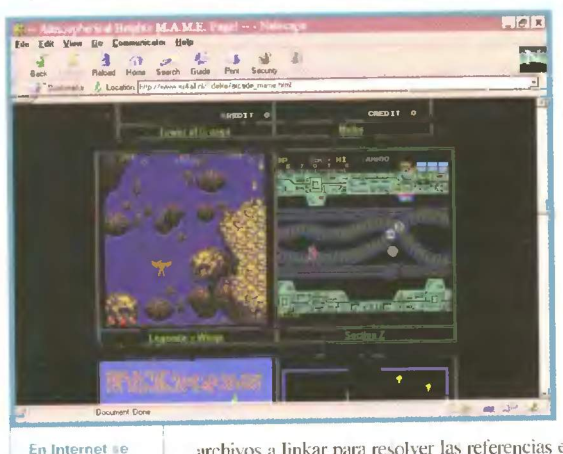
OPCIONES DE LINKADO

El proceso de linkado puede ordenarse o bien a través del GCC o bien llamando directamente al linkador Ld, aunque esto último no se recomienda. En términos generales, el usuario creará el ejecutable directamente con GCC, prescindiendo de Ld, y de igual manera lo más corriente será crear el ejecutable directamente a partir de los fuentes con una sola orden a GCC.

DJGPP V.2.1

Cuando decimos "linkar mediante GCC" nos referimos al hecho de que lo que hace GCC es llamar por su cuenta al linkador Ld. Esto quiere decir que dentro de dicha orden tendremos que especificar las opciones para el linkador y el orden en que van a tomarse los archivos objeto y las librerías para efectuar el proceso de linkado. Naturalmente ciertos programas como hola.c no necesitan que se les especifiquen opciones de linkado ni librerías, pero lo cierto es que automáticamente GCC tratará por su cuenta una librería (libc.a) para crear el ejecutable. En caso de que sean precisas más librerías usaremos el comando "-l" añadiendole el nombre de la librería. Hay que tener en cuenta que las librerías se encuentran en el subdirectorio LIB y sus nombres empiezan siempre por lib, aunque este prefijo no se emplea en la orden "-l". Así, por ejemplo, para ordenar a GCC que procese una librería del usuario llamada libyuyu.a, habríamos de emplear la orden "-lyuyu" (y también sería preciso incluir a libyuyu.a en el subdirectorio de librerías).

Otro detalle muy importante que se debe recordar es que el linkador procesará los archivos objeto y las librerías en el orden en que se hayan colocado dentro de la línea de órdenes. O sea que en "kk.o -lpaka maskk.o", la librería paka.a sería procesada después que kk.o y antes que maskk.o. Ello tiene su importancia puesto que maskk.o podría referenciar a funciones de paka.a. Unas funciones que probablemente no serían consideradas, ya que el linkador solo efectúa una pasada sobre los archivos a linkar para resolver las referencias externas (las referencias a funciones de librerías). Por esta razon debemos colocar siempre las órdenes "-l" de las librerías después de todos los archivos .o a linkar. Ahora una cosa más; para usar las clases de C++ necesitaremos invocar a la librería libgpp.a y también habrá que referenciar a la plantilla de clases estándar de C++, lo que haremos con la librería libstdcx.a. Por último, en caso de que debamos usar funciones matemáticas especiales (que no estén incluidas dentro de libc.a) usaremos la librería libm.a. Estas librerías deberán ser referenciadas siguiendo un orden preciso que es:



En Internet se pueden encontrar miles de programas shareware y freeware creados con DJGPP y sus herramientas.

gcc -o main.exe main.cc sub.cc -lgpp -lstdcx -lm

Ahora supongamos que tenemos una librería propia llamada libmia.a que deseamos emplear. Esta deberá ser colocada antes que las demás:

gcc -o main.exe main.cc sub.cc -lmia -lgpp -lstdcxx -lm

También es conveniente recordar que, si por la razón que sea, cambiáis el nombre del subdirectorio de las librerías, entonces deberéis asimismo cambiar la variable LIBRARY_PATCH en el fichero DJGPP.ENV (en el apartado GCC) para que apunte a la nueva posición. De lo contrario las librerías no serán halladas por GCC. Por último conviene reseñar que este cambio no tendrá efecto si invocamos directamente al linkador Ld ya que esta variable solo es conocida por GCC. En este caso deberemos usar la opción -L añadiéndole la ruta con la posición de las librerías. GCC también acepta esta opción así que podremos invocar sin problemas a librerías situadas en distintos directorios. Veamos un ejemplo:



La calidad de las librerías gráficas elegidas en la instalación afecta al resultado de nuestra creación

gcc -o programa.exe programa.c -L.c:\fuentes\library -lm

Ahora veamos otro caso: puede ocurrir que, por la razón que sea, nos interese compilar un fuente pero no linkarlo. En este caso emplearemos la opción -c que le dirá a DJGPP que puede ensamblar y/o compilar, pero que se olvide del proceso de linkado. Después de esto el resultado será un fichero .o (objeto) por cada fuente. Más tarde podríamos crear un ejecutable con, por ejemplo, esta línea: **gcc -o programa.exe programa.o -lgpp -lstdcxx**

OTRAS OPCIONES DE GCC

Los comandos posibles que admite GCC son muchísimos (¿quizá cientos?) pero no todos tienen la misma utilidad. Seguidamente echaremos un vistazo a los más corrientes.

-o: Coloca la salida (output) producida por el compilador en el fichero cuyo nombre se indica a continuación. Esta opción no tiene nada que ver con "O" (optimizaciones), así que cuidado. Ejemplo: **Gcc -o hola.exe hola.c**

-ansi: desactiva todas las características del GNU C que son incompatibles con el estándar ANSI tales como asm e inline. También se desactivan las macros como "unix" y "vax" que identifican el tipo de sistema que sirve como soporte.

-fsyntax-only: chequea el código para comprobar errores sintácticos pero no hace nada más.

-w: inhibe todos los mensajes de advertencia (warning). Como ya sabéis, este tipo de mensajes no señala errores lo bastante graves como para fastidiar la creación del ejecutable, aunque sí marca inexactitudes que podrían ser la causa de un mal funcionamiento.

-werror: hace que todos los warnings sean tratados como errores por el compilador.

-v: imprime datos acerca del proceso de compilación, lo cual puede resultar útil para saber en

DJGPP V.2.1

qué punto del proceso se han producido los problemas.

-g: produce información simbólica en el código resultante de la compilación a fin de que el ejecutable pueda ser trazado paso a paso posteriormente desde un depurador. Podremos usar esta opción aunque las optimizaciones estén activadas, lo que será una buena forma de comprobar cómo funcionan muchas de las optimizaciones manuales. La opción -g puede ser usada para crear código simbólico de varios formatos y esto es importante ya que puede ampliar el espectro de depuradores que podremos utilizar para trazar el código.

Veamos algunas opciones posibles: -ggdb produce información en el formato nativo si este está soportado; -gstabs produce el formato stabs; -gxcoff crea el formato coff, etc. Además, adicionalmente es posible especificar el nivel de información que -g debe producir. Con -g1 produciremos el nivel mínimo y con -g3 el máximo (el -g2 es el empleado por defecto). Nota: solamente en el apartado de las opciones de control de la información simbólica que se ha de producir, GNU ofrece unas 40 opciones.

MAKE

DJGPP también dispone de Make. Esta utilidad que conocen (o recuerdan) la mayoría de los programadores, salvo quizás los que trabajan con entornos integrados, nos servirá para realizar la compilación de proyectos complejos compuestos por muchos fuentes. Con Make podremos controlar las opciones y las condiciones de compilación de cada fuente mediante un fichero al que llamaremos makefile en el que se especificarán todas estas cosas. Gracias a Make, además, no será preciso compilar todos los ficheros antes del linkado, a menos que todos ellos hayan sufrido cambios. El funcionamiento del Make de GNU no difiere en general, pues, con respecto al de los makes de otros compiladores.

Cuando, por ejemplo, alteremos un fichero .h de nuestro proyecto, cada uno de los archivos de C que lo referencie será compilado de nuevo. Y de nuevo, como en el caso de otros compiladores, la compilación de cada fichero de C producirá un fichero objeto y asimismo el cambio en cualquier fuente hará necesario linkar todos los archivos-objeto (que, repetimos, en GNU tienen la extensión "o"). En cuanto a la estructura del Makefile, esta es prácticamente idéntica a la de los ficheros make que la mayoría de los programadores ya conocen, conservándose incluso detalles de sintaxis tales como la barra inclinada para indicar que se van a continuar especificando ficheros en la línea siguiente.

El lector podrá comprobar esto en la figura siguiente en la cual se incluyen un par de trozos del larguísimo Makefile de POV (como sin duda sabrá el lector, POV es un trazador de rayos de libre distribución y uso cuyos fuentes pueden ser libremente trasteados por el usuario para crear su propia versión de este programa. Junto a estos fuentes se adjuntan los archivos makefile para compilar POV con los compiladores Watcom, Borland o DJGPP, lo cual ya de por sí dice mucho acerca de la popularidad y potencia de DJGPP).

Make es una de las utilidades mejor documentadas del paquete GNU y el usuario podrá encontrar una completa introducción sobre esta herramienta, así como detalles y ejemplos de todas las opciones posibles. Por nuestra parte no entraremos en detalles sobre Make por dos razones: la primera, porque es una herramienta compleja cuyo estudio detallado ya sería más que suficiente para llenar de contenido un librito como este, y la segunda porque Make no es más que una de las múltiples opciones posibles de las que puede disfrutar el usuario de DJGCC. Además existió hace algún tiempo una acalorada discusión entre los programadores acerca del uso de Make y sus sustitutos. La causa fue, como ya puede imaginar el lector, la aparición de entornos integrados que incluían opciones de sobra para construir proyectos complejos sin necesidad de usar un método tan artesanal como es Make. Contra esto, los partidarios de Make aducían la flexibilidad de esta herramienta y llamaban comodones a los usuarios que preferían los entornos integrados. Estos últimos a su vez replicaban llamando fósiles a los make-adeptos por su negativa a modernizarse.

Nota del autor: aunque los propios archivos fuente de POV demuestran que Make sigue siendo una herramienta vivita y coleando, útil para crear auténticos programas de categoría profesional, creo que se trata de un sistema que tenderá a desaparecer con el tiempo a consecuencia de la cada vez mayor potencia, sencillez y comodidad de los entornos integrados. Sin embargo es probable que el Make de GNU en concreto se mantenga en el candelero aún durante mucho tiempo ya que los entornos integrados de GNU son relativamente recientes y llevan, pues, algo de retraso con respecto a los de otros compiladores comerciales. En fin, llegados aquí quizá debiera admitir que yo prefiero los entornos integrados. De hecho me pasé a ellos hace ya tanto tiempo que apenas recuerdo el manejo de Make.

PARTES DEL MAKEFILE DE POV

```
...
...
CFLAGS=-O3 -m486 -funroll-loops  (aquí están las opciones de optimización para compilar)
...
...
(sigue el rebaño de objs que componen POV)
POVOBJS = atmosph.o bbox.o bcyl.o bezier.o blob.o boxes.o \
  bsphere.o camera.o chi2.o colour.o cones.o \
  console.o csg.o discs.o express.o fractal.o gif.o \
  gifdecod.o halos.o hcmplx.o hfield.o iff.o image.o \
  lathe.o lbuffer.o lighting.o matrices.o mem.o \
  mesh.o msdostxt.o msdosvid.o normal.o objects.o \
  octree.o optin.o optout.o parse.o parstxtr.o \
  pattern.o pgm.o pigment.o planes.o png_pov.o \
  point.o poly.o polygon.o polysolv.o povray.o ppm.o \
  prism.o quadrics.o quatern.o rad_data.o radiosit.o \
  ray.o render.o sor.o spheres.o super.o targa.o \
  texture.o tokenize.o torus.o triangle.o truetype.o \
  txttest.o userio.o vbuffer.o vesavbe.o vlbuffer.o \
  warps.o pmlite.o pmpro.o vflat.o _pmlite.o _pmpro.o _vflat.o
...
...
povray.exe : $(POVOBJS)   (el exe se compone de los objs de arriba)
            gcc -o povray @msdosgcc.ln1
            strip povray
            coff2exe povray
            del povray.
...
...
```

{capítulo 3}

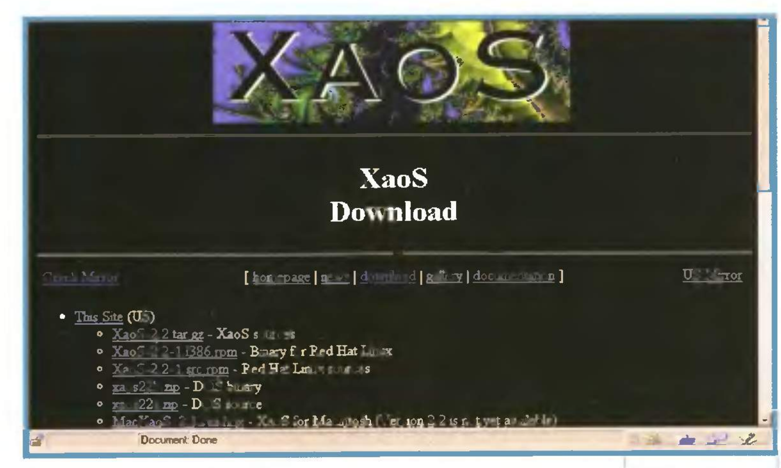
PROBLEMAS AL CREAR EL EJECUTABLE

Aquí veremos algunos de los muchos problemas con los que puede tropezarse el usuario a lo largo del camino que debe recorrerse para crear un ejecutable.

PROBLEMAS CON LA COMPILACIÓN Y EL LINKADO

Naturalmente es imposible prever todos los problemas que pueden presentarse cuando se está intentando crear un ejecutable con DJGPP, pero puede ser ilustrativo citar algunos de los más frecuentes:

- A veces algunos usuarios se encuentran con el caso de que el mero intento de invocar a GCC provoca el mensaje de error siguiente "Load Error: No Dpmi". Esto significa, tal y como dice el mensaje, que no tenemos instalado un programa que ofrezca servicios DPMI a GCC. En otras palabras: que necesitamos instalar un servidor DPMI cualquiera, por ejemplo, Cwsdpmi, que es de libre distribución (y que se adjunta en el CD). Otro problema parecido con el que se topan de vez en cuando los usuarios despistadillos es el que ocurre cuando las aplicaciones compiladas con GNU versión 2 se cuelgan, sin que esto suceda con las generadas con la versión anterior del compilador. Esto sucederá porque el programa DPMI que está instalado estará dando problemas (un caso muy problemático, según algunos FAQ, es el de Novell Nwdos y también se menciona a Qdpmi de Quarterdeck). En estos casos el usuario deberá desactivar el DPMI problemático y usar Cwsdpmi. En la práctica esto quiere decir que deberíamos distribuir Cwsdpmi con nuestras aplicaciones o, al menos, indicar su existencia y dónde conseguirlo en nuestra documentación (Nota: la libre distribución de Cwsdpmi está permitida requiriéndose únicamente que enviemos un mensaje de correo electrónico al autor agradeciéndole la existencia de este programa).
- La versión 2.0 tenía algunos bugs que se han corregido en la versión 2.1 y de la misma manera errores de la última versión pueden verse corregidos en próximos parches publicados en el web de Delorie o en ftp.simtel.net (en el directorio pub/simtelnet/gnu). Nota: que se hable de errores de código no debe hacernos menospreciar a DJGPP. Estas cosas ocurren con casi todas las nuevas versiones de cualquier programa grande y complejo. En general a la aparición de una nueva versión de cualquier programa suele seguir otra que corrige los errores de la anterior.
- Si por casualidad alguien estuviera usando la versión 2.6.0 de GCC, que tome nota de que dicha versión tendía a colgarse con las opciones de optimización (la versión incluida en el CD es la 2.7.2, que es mucho menos proclive a este tipo de desastres).
- Otro caso curioso: los gurús de DJGPP aseguran que cuando el compilador devuelve un mensaje como "Unknown file" (fichero desconocido) o cuando la ejecución del ejecutable obtenido nos da el mensaje "Not Coff", ello puede deberse a que un virus ha infectado a GCC. Al parecer la mayoría de los virus cuando atacan a GCC se instalan en la zona de éste donde se tratan cuestiones vitales relacionadas con la generación de código, lo cual da como resultado dichos mensajes. Así que ya sabéis, si los nombres de los ficheros están correctos en la línea de compilación y el mensaje "unknown file" aparece: ¡poned en marcha un antivirus enseguida!



Al descargar librerias para DJGPP a traves de Internet hay que cerciorarse de que sean las versiones para MS-DOS

- Puede ocurrir a veces que la compilación fracase sin que parezcan existir razones claras para ello. En este caso puede resultarnos útil el comando -v que imprimirá un texto (por defecto en la pantalla) en el que se mostrarán los detalles de la compilación. Por otro lado una posible respuesta al problema puede estar en la falta de RAM (según se advierte en la documentación, la falta de memoria puede hacer que GCC corra muy lentamente e incluso que se cuelgue). Para comprobar esto podéis intentar implementar la misma instalación en el ordenador de algún amiguete y hacer la misma prueba. Si el fallo no aparece y las instalaciones son similares, esta es la causa más probable.
- Un problema menor relacionado con la opción -v es que tiende a generar una enorme masa de datos que se mostrarán por defecto en la pantalla a gran velocidad, lo que nos impedirá leer los textos. Para poder estudiar estos datos con más tranquilidad lo mejor es redirigir la salida a un fichero, para lo cual podemos emplear el programa redir (escrito por Delorie) que se adjunta con GCC. Por ejemplo, en la línea:

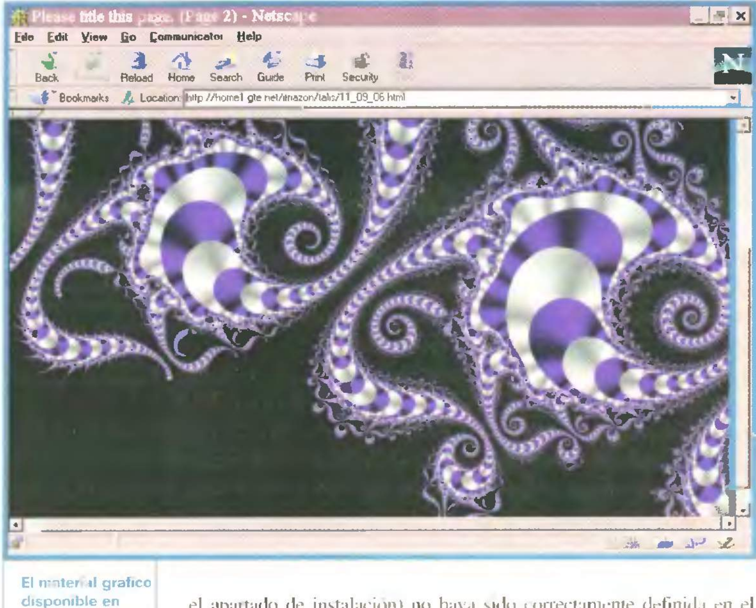
redir -o gcc.log -eo gcc -v ...

la salida de -v se redirige al archivo gcc.log. La opción -o de redir es la que ordena que la salida se guarde en el archivo cuyo nombre se indica a continuación. En cuanto a -eo, especifica que los errores sean redirigidos a la salida (para ver más detalles sobre el funcionamiento de redir, simplemente llamad al programa sin darle parametros).

- Una manera bastante tonta de meter la pata consiste en equivocarse con el Make que se está usando (si es que estamos empleando Make, claro). Casi todos los programas de este tipo se lla-

man Make y a menudo suele suceder que el usuario dispone de más de un compilador. Incluso en el caso de que estuviéramos empleando el Make de la anterior versión de GNU tendríamos problemas porque los programas de dicha versión intentarían llamar a go32 para funcionar. Estos choques se producen porque la filosofía de la nueva versión ha cambiado un poco. Los ejecutables ya no deben llamar a go32 porque parte del código de este programa está incluido en ellos.

⊙ Si el compilador genera un mensaje diciendo que no puede encontrar ficheros de cabecera y/o librerías, esto puede deberse a diversas causas. Una de ellas puede ser que la variable DJGPP (ver el apartado de instalación) no haya sido correctamente definida en el archivo AUTOEXEC.BAT o que esté apuntando a donde no debe. También puede ocurrir que, simplemente, el AUTOEXEC.BAT esté mal escrito; Por ejemplo, una manera habitual de meter la pata consiste en poner "DJGPP " en vez de "DJGPP", lo cual no es lo mismo (sobra el espacio en blanco en el primer caso). Otra posibilidad es que el usuario haya cambiado la posición del archivo DJGPP.ENV, que debe permanecer en el directorio raiz de DJGPP. Por último, si se está ejecutando DJGPP dentro de una ventana de Windows 95 o NT o de cualquier otro entorno que soporte nombres largos, deberá tenerse presente que los nombres de directorios dentro de DJGPP.ENV no deberán exceder los 8 caracteres de largo (excluyendo los 3 del sufijo). También, en casos raros, puede suceder que el editor de textos que utilicemos se encargue de estropear DJGPP.ENV. Otras causas pueden ser que el usuario haya renombrado a GCC (porque en ese caso DJGPP.ENV ya estaría referenciando al compilador) o que el valor Files de config.sys sea demasiado bajo (al menos debería ser de 15).



El material grafico disponible en ciertas librerias de C es realmente extenso y variado. Internet es la mejor fuente para conseguirlas.

⊙ En la versión 2.5.2 de Binutils el linkador Ld tenía un bug que ha sido resuelto en la versión 2.7 que adjuntamos en el CD, pero aún así puede ocurrir que de vez en cuando aparezca un mensaje al linkar diciendo que no se han podido leer todos los símbolos o que no hay memoria suficiente. En estos casos los gurús recomiendan dividir las librerías grandes en otras mas pequeñas.

⊙ Algunos de los problemas pueden aparecer si instalamos la versión 2 sobre la anterior. Por ello lo mejor es borrar la anterior o, al menos, renombrarla para que no choque con la nueva. Existen muchas más causas posibles de error pero casi todas suelen deberse a fallos de instalación y configuración. Aunque... ¿qué podemos hacer si seguimos sin identificar correctamente la causa del error?

¡SOCORRO!

⊙ Casi con toda seguridad cualquier problema ante el que nos topemos ya le habrá sucedido antes a otro usuario. Por ello conviene saber que, si se cuenta con una conexión a Internet, no será mala idea mirar en los archivos de correo de Delorie (en la dirección http://www.delorie.com/djgpp/mail-archives). Casi con toda seguridad allí podremos estudiar la respuesta que dio algun gurú a alguien con un problema idéntico o muy parecido al nuestro. El problema será únicamente encontrar el mensaje con la respuesta, ya que puede haber miles almacenados. Para ello podremos efectuar la búsqueda empleando el buscador de la pagina o examinar los mensaje de un periodo de tiempo concreto. En general lo mas cómodo suele ser usar el buscador dándole las palabras clave relacionadas con nuestro problema (Nota del autor: cuando entré aquí, la página indicaba que había más de 56.500 mensajes en la base de datos; entonces utilicé la frase "unknown file" en el buscador y éste me presentó una extensa lista de mensajes; pinché sobre una y...). Según se afirma, el índice de mensajes se actualiza cada 24 horas.

⊙ Otra posibilidad es pedir socorro a alguno de los gurús de DJGPP (Nota: el autor de este artículo carece de la suficiente experiencia con DJGPP como para atreverse a ostentar dicho título). Afortunadamente existen al menos dos sitios donde uno puede enviar sus preguntas (aunque se recomienda probar antes en la base de datos de Delorie. A los gurús no les gusta responder mil veces a la misma pregunta). Estos dos lugares son el grupo de noticias de comp.os.msdos.djgpp y djgpp@delorie.com. En ambos casos se pide al usuario que emplee el siguiente sistema:

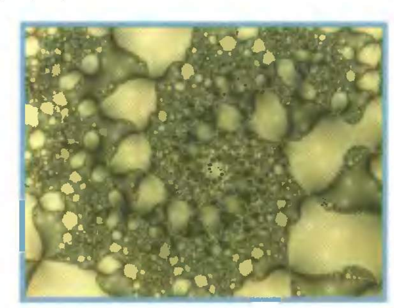
a) Desde el DOS ordenar set > environ.lst
b) Desde el subdirectorio BIN invocar a go32-v2 y grabar su salida a un fichero.
c) Enviar a alguna de las dos direcciones indicadas más arriba un mensaje de correo electrónico describiendo el problema junto con el archivo environ.lst, la salida de go32-v2, los archivos AUTOEXEC.BAT y CONFIG.SYS, y la salida de GCC durante la compilación utilizando -v (si GCC no termina la compilación).

Segun se dice, la respuesta puede tardar tan solo dos o tres días.

DJGPP V.2.1

capítulo 4
ENSAMBLADOR BAJO GNU

A pesar de la cada vez mayor velocidad de los equipos y también de la cada vez mayor rapidez del código que generan los compiladores, la mayoría de los programadores acaban interesándose tarde o temprano por la programación en bajo nivel, esto es; por el ensamblador. En efecto, a pesar de que la programación en ensamblador es algo arcaico, difícil y poco dado a la estructuración, tarde o temprano casi todos los programadores coquetean con ella. ¿Por qué? Pues porque inevitablemente la competencia nos empuja a crear un código más rápido que el del vecino y así hoy día el ensamblador sigue siendo el lenguaje que marca la diferencia en determinadas rutinas cuya máxima velocidad es vital. Por esta razón casi todos los compiladores de C incluyen alguna sentencia parecida a asm para insertar instrucciones de ensamblador dentro de los fuentes de C (aparte, por supuesto, de la posibilidad de linkar objetos procedentes de fuentes en ensamblador).



LA CUESTIÓN DE LA SINTAXIS

Para aquellos lectores que hayan empleado la versión anterior de DJGPP y en particular para aquellos amantes del ensamblador que enviaron algunas cartas a PCmanía al respecto hace tiempo, anticiparemos que desgraciadamente la sintaxis que emplea el ensamblador de DJGPP sigue siendo, como en la versión anterior de este paquete, la de AT&T en vez de la de Intel, que es la que todos conocemos (ay, ay). Seguidamente en atención a los nuevos adeptos de DJGPP explicaremos las diferencias de sintaxis más importantes entre ambos formatos. Por tanto, quienes ya conozcáis este tema podéis obviar estos apartados tranquilamente (Nota del autor: quiero aclarar que no he probado muchos de los ejemplos que aquí veis y que han sido tomados de diferentes FAQ sobre este tema; aunque, según creo, no se han producido cambios de formato con respecto a la primera vez que tratamos este asunto para la versión anterior de DJGPP).

Las diferencias principales entre ambos formatos son:

1) En la sintaxis de AT&T se usa el orden opuesto al que emplea el formato de Intel para los operandos destino y fuente. O sea que en DJGPP tendremos que especificar primero el operando fuente y luego el de destino (tal como sucede en los ensambladores para la serie 68000 de Motorola).

2) Los registros van precedidos por el carácter %.

3) Los operandos inmediatos son precedidos por el carácter $ y se supondrá que los valores serán decimales (a menos que se emplee el formato 0x).

4) La longitud de los operandos de memoria debe ser especificada añadiendo un carácter al final del mnemónico. Los caracteres permitidos son "b" (8 bits), "w" (16 bits) y "l" (32 bits). En la figura podéis ver unos ejemplillos comparativos entre la sintaxis de AT&T y la de Intel.

5) En los casos de direccionamiento indirecto se usarán paréntesis para encerrar al término. Por ejemplo (%esi).

6) En casos como "movsx ecx,ax" será preciso indicar las longitudes de los dos operandos quedando la instrucción tal como sigue "movswl %ax, %ecx". Nota: esto es inútilmente redundante ya que la longitud ya se está indicando con los propios registros y es, además, lioso, pero así es la sintaxis de AT&T).

7) Por alguna razón no se admite que los prefijos "rep" permanezcan en la misma línea que el resto de la instrucción, quedando una única instrucción dividida en dos.

8) Las referencias indirectas a memoria también sufren cambios en el formato de AT%T como podéis ver en la figura que contiene los ejemplos comparativos correspondientes.

9) Las instrucciones de salto condicional sólo podran ser usadas para desplazamientos de 8 bits, lo cual nos obligará a cambiar un poco nuestros viejos fuentes.

10) Para el caso de los saltos "largos" como "Call far" o "Jump far", los mnemónicos deberán ir precedidos por el carácter "l" como puede verse en los ejemplos de la figura dedicada a los saltos largos.

EJEMPLOS COMPARATIVOS ENTRE LA SINTAXIS DE AT&T Y LA DE INTEL

FORMATO DE AT&T	FORMATO DE INTEL
movw %bx, %ax	(mov ax, bx)
xorl %eax, %eax	(xor eax, eax)
movw $1, %ax	(mov ax,1)
movb X, %ah	(mov ah, byte ptr X)
movw X, %ax	(mov ax, word ptr X)
movl X, %eax	(mov eax, X)

EJEMPLOS COMPARATIVOS ENTRE LA SINTAXIS DE AT&T Y LA DE INTEL SOBRE REFERENCIAS DE MEMORIA

FORMATO DE AT&T	FORMATO DE INTEL
movl 4(%ebp), %eax	(mov eax, [ebp+4])
addl (%eax,%eax,4), %ecx	(add ecx, [eax + eax*4])
movb $4, %fs:(%eax)	(mov ls:eax, 4)
movl array(,%eax,4), %eax	(mov eax, [4*eax + array])
movw array(%ebx,%eax,4), %cx	(mov cx, [ebx + 4*eax + array]

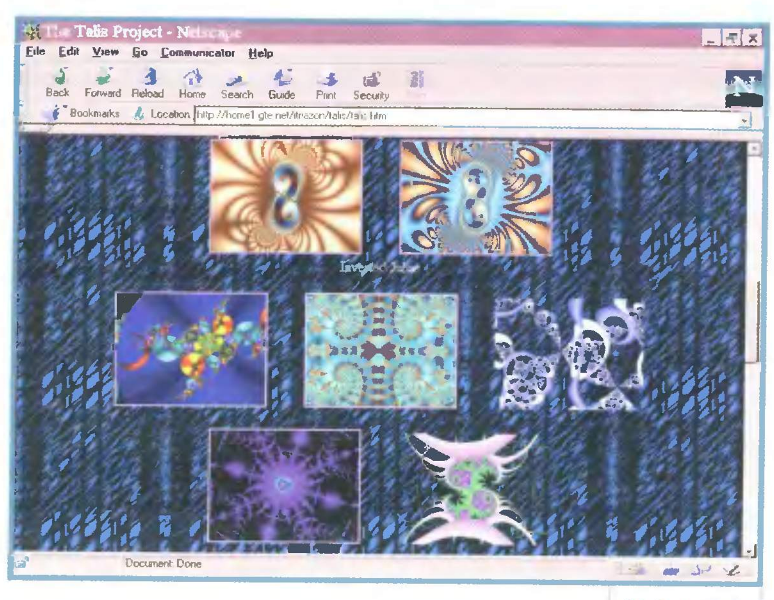
EJEMPLOS COMPARATIVOS DE SALTOS LARGOS EN AMBOS FORMATOS

FORMATO DE AT&T	FORMATO DE INTEL
lcall $S,$O	(call lar S-O)
ljmp $S,$O	(jump lar S:O)
lret $V	(ret far V)

EJEMPLOS COMPARATIVOS DE CAMBIOS DE NOMBRE CON RESPECTO AL FORMATO DE INTEL

FORMATO DE AT&T	FORMATO DE INTEL
cbtw	(cbw)
cwtl	(cwde)
cwtd	(cwd)
cltd	(cdq)

DJGPP V.2.1



Muchos programas de generación de fractales han sido creados, linkados y compilados con las diversas versiones de GCC y DJGPP.

11 Algunos mnemónicos sufrirán cambios drásticos como se refleja en la última de las figuras incluidas en esta página.

12 Según se afirma en la documentación del programa, para el caso de las multiplicaciones (mul e imul), las instrucciones con dos operandos no expandirán el resultado en el registro DX o EDX. Habrá que utilizar, por lo tanto, aquellas instrucciones que empleen un sólo operando, como por ejemplo "imul %ebx", para conseguir que la expansión del resultado se efectúe de un modo satisfactorio.

CÓDIGO ENSAMBLADOR EN FUENTES DE C

El formato usado por DJGPP para incluir líneas en ensamblador en los fuentes de C es:

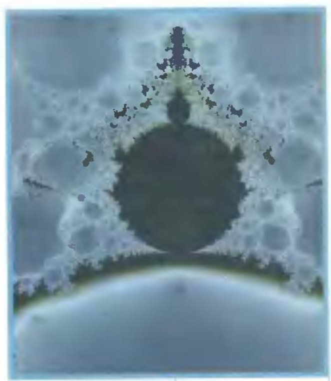
```
__asm__(instrucciones
     ...
     ...
     : salidas
     : entradas
     : registros-modificados);
```

El primer campo estará compuesto por instrucciones de ensamblador de las que debe haber solamente una por línea. Los dos campos siguientes son las variables de C que se tomarán como salidas y entradas del trozo de código en ensamblador. Cada uno de estos campos comenzará con el carácter ":" al que seguirán las variables separadas por comas. En caso de que no vaya a haber por ejemplo variables de salida se incluirá el carácter ":" y se dejará vacía la línea. Finalmente, en el último campo y empleando el mismo formato, indicaremos las variables de C que van a resultar modificadas por el código ensamblador. Por supuesto, en el caso de que no haya variables de entrada ni de salida, las líneas con el carácter ":" no serán colocadas. El siguiente es el ejemplo más sencillo posible:

```
__asm__("
   pushl %eax\n
   movl $1, %eax\n
   popl %eax"
  );
```

El ejemplo anterior no sirve absolutamente para nada salvo para mostrar un caso de sintaxis correcta (se guarda eax en el stack, se pone eax a 1 y finalmente se recupera el valor anterior de eax, con lo que todo queda como antes). El ejemplo siguiente al menos tiene algun efecto:

```
int i = 0;
  __asm__("
   pushl %%eax\n
   movl %0, %%eax\n
   addl $1, %%eax\n
   movl %%eax, %0\n
   popl %%eax"
   :
   : "g" (i)
  );
```



Las posibilidades de la programación en ensamblador son enormes, aunque quizá el modulo contenido en DJGPP presente demasiados inconvenientes.

En este ejemplo no se pone ningún resultado en ninguna variable de salida, por tanto el campo de salidas se marca pero queda vacío y, al no haber ninguna variable modificada, el último campo queda también vacío por lo que podemos eliminarlo. La variable i del fuente de C se ha empleado como entrada y es referenciada como %0 dentro del código ensamblador. Este último valor indica el orden en que ha sido declarada la variable en los campos segundo y tercero, por lo cual es fácil imaginar que, en caso de que hubiera una variable de salida, esta pasaría a ser %0, mientras que la de entrada sería llamada %1. Además (para acabar de liar aún más la cosa) todos aquellos registros que pasen a contener valores de variables de C serán indicados con "%%" en vez de con "%". Al llegar aquí el sufrido programador de ensamblador de Intel estará ya de seguro echando humo, pero aún hay más: las variables dentro de los tres campos finales estarán encerradas entre paréntesis y precedidas por un carácter entrecomillado que normalmente será "g". Esto parece significar que se deja a elección del compilador el registro donde se va a leer el valor de la variable de C. Otras posibles opciones son "r" (leer en cualquier registro disponible), "a"

tax/eax), "b" (bx/ebx), "c" (cx/ecx), "d" (dx/edx), "D" (di/edi), "S" (si/esi)... En general, según parece, suele ponerse "g", lo cual deja mano libre a GCC para que intente optimizar incluso nuestras propias líneas de ensamblador. No obstante, puede que esto no siempre nos haga gracia (a fin de cuentas si incluimos líneas de ensamblador será porque queremos controlar muy bien el código máquina que va a emplear el programa) y por esa razón existe la sentencia "__asm__ __volatile__" que emplearemos en sustitución de "__asm__". Esta nueva palabreja indica a GCC que desista de intentar optimizar nuestras líneas de ensamblador.

NOTA DEL AUTOR

Existen aún muchas más cuestiones por explicar sobre el uso del ensamblador en línea pero, ¿para qué continuar con este tema? Lo cierto es que casi cualquier programador de ensamblador que haya leído estas líneas renunciará casi con toda seguridad a introducir ensamblador con DJGPP porque, admitámoslo de una vez, las reglas son redundantes y farragosas y son, con mucho –en mi modesta opinión– lo peor que podemos sacar como conclusión de DJGPP. Creo, además, que la mayoría de los lectores también fueron de la misma opinión cuando, hace ya algún tiempo, se publicó en la revista PCmanía la anterior versión de DJGPP y todos realizaron sus propios experimentos. ¡Muchos me pidieron a gritos que les dijese dónde podían encontrar diferentes utilidades conversoras del formato de Intel a la nefasta sintaxis de AT&T! Entonces no pude hacerlo (aunque sí tenía intención) pero ahora, afortunadamente, cuento con una maravillosa conexión a Internet y sí que puedo dar algunas respuestas, así que, queridos maníacos y amantes del ensamblador, abrid muy bien las orejas que esto os puede interesar enormemente:

1| Existe un ensamblador de libre distribución llamado Nasm del que además se distribuyen los fuentes, que hallaréis en http://www.cryogen.com/Nasm. Nasm es capaz, según parece, de tomar fuentes de ensamblador en el formato de Intel y producir codigo COFF compatible con DJGPP. No he trasteado aún personalmente con él pero todo cuanto he oído es muy bueno.
2| Presuntamente el conocido MASM puede producir código COFF a partir de nuestros fuentes (aunque yo nunca he conseguido verificar esto).
3| Existe una utilidad llamada Ta2as (en ftp://x2ftp.oulu.fi, en el directorio pub/programming/convert) que presuntamente convierte codigo de Intel al formato de AT&T. La última versión en el momento de escribir esto estaba guardada en ta2asv08.zip.
4| También hay otra utilidad llamada O2c (que puede ser hallada en el mismo lugar y directorio que la anterior) que (de nuevo presuntamente) convierte archivos .obj y librerías entre TASM y DJGPP aunque, todo hay que decirlo, el FAQ de Eli Zaretskii menciona algún problemilla sobre ella. La última versión que vi estaba en el archivo o2cv10.arj.
5| Para quienes quieran estudiar a fondo el formato de AT&T, los archivos de mensajes de Delorie incluyen preguntas acerca de la conversión entre los dos formatos. Aunque lo más rápido será probablemente estudiar el tutorial de Brennan "Bas" Underwood en http://www.rt66.com/~brennan/djgpp.
6| Nota: Gas, que así se llama el ensamblador de GNU, no parece un programa demasiado fiable (usad mejor el código en línea).

capítulo 5
MISCELÁNEA DE TEMAS SOBRE DJGPP

En el presente capítulo se exponen tan sólo algunos de los incontables temas que sobre la programación con DJGPP pueden interesar al usuario. Hemos dado por supuesto que el lector está familiarizado con lo que son los debuggers o depuradores de código, la programación en modo protegido, los selectores, etc., ya que todos estos temas se han tratado en uno u otro momento en PCmanía y además son bien conocidos para la mayoría de los programadores.

DEPURACIÓN DE PROGRAMAS COMPILADOS CON DJGPP

Ante todo hay que recordar a todos los usuarios utilizar la opción "-g" en la línea de órdenes de GCC y especificar el nivel de información simbólica que queremos. También hay que evitar el empleo de la opción "-s" del linkador, ya que ésta elimina toda la información simbólica que luego necesitarán los depuradores. Posteriormente, una vez que tengamos el ejecutable, dispondremos de las diferentes opciones que vamos a explicar a continuación:

1| Podemos emplear RHIDE, el entorno IDE estilo Borland creado por Robert Hoehne. Este programa incluye un debugger de fuente basado en código GDB.
2| Otra opción es Rhgdb. Este programa es parte de RHIDE y tiene una interfaz de estilo Turbo Vision (Nota: el autor de estas líneas no ha podido probar aún este programa).
3| FSDB (Sally Full Screen Debugger) es el depurador de código que se adjunta con Djdev. Se trata de un debugger con una interfaz muy parecida a la de Turbodebugger pero que, a diferencia de éste, no puede depurar código fuente, sino tan sólo código máquina. Entre sus opciones, soporta breakpoints y tiene otras características muy interesantes, pero desgraciadamente no permite estudiar con facilidad estructuras complejas de datos. En nuestra opinión, aunque interesante, la imposibilidad de usarlo con fuentes de C o C++ le resta mucha utilidad. Usad la tecla F1 para solicitar ayuda.
4| El debugger oficial de GNU es el GDB, un potente depurador de fuentes. Pero desgraciadamente carece de un entorno gráfico y funciona a través de órdenes impartidas en línea. Esto fastidiará extraordinariamente a aquellos usuarios que -como vuestro seguro servidor- estén acostumbrados a los entornos integrados.
5| Por último tenemos a Edebug32, que es quizá el programa más simple de todos los aquí comentados y cuya utilidad es, pues, bastante limitada.

PROGRAMACIÓN DE BAJO NIVEL

La expresión "bajo nivel" no se refiere únicamente a la programación de ensamblador sino a otros muchos asuntos relacionados con la programación directa del hardware, el uso de los servicios DPMI, las interrupciones del DOS o a cuestiones como, por ejemplo, intercambiar datos entre la memoria convencional (el primer Megabyte) y nuestro programa DJGPP (situado en la memoria extendida). En este último caso hay que recordar que no podremos acceder tranquilamente a dicha memoria convencional y que para ello tendremos que emplear alguno de los siguientes artificios:

1 | Usar el selector que DJGPP tiene preparado para que el programador pueda acceder a la memoria del primer Mega. Este selector es _dos_ds, está definido en go32.h y tiene su base en la dirección absoluta 0 y su límite en la cima del primer Mega. Esta es una posible forma de acceso a cualquier dirección de la memoria convencional, como por ejemplo la memoria de vídeo, sin que tengamos que molestarnos en crear un selector.



2 | Otra posibilidad es, por supuesto, crear nuestro propio selector y usarlo en vez de _dos_ds. En el listado podéis ver un ejemplo creado por Bill Currie (y ladinamente tomado del FAQ de Zaretskii) en el que se crea un selector para acceder a los 64 Kilobytes de la dirección de la memoria de vídeo en modo texto (0xB800:0000).

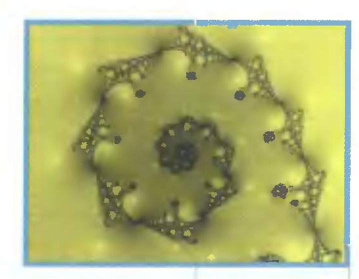
Ejemplo de creación de un selector propio para una zona de memoria del primer Mega con DJGPP:

```
int TxtVRAMSetupSelector (void)
{
  static char selectorData[8] = {
   0xff, 0xff, 0x00, 0x80,
   0x0b, 0xf3, 0x40, 0x00
  };
  int screenSelector = __dpmi_allocate_ldt_descriptors (1);
  if (__dpmi_set_descriptor (screenSelector, selectorData) < 0)
   abort ();
  return screenSelector;
}
```

El problema de los selectores de acceso al primer Mega tiene varias soluciones en DJGPP, pero ninguna de ellas resulta precisamente sencilla para no iniciados.

Entre las ventajas de este método está el hecho de que uno puede definir el límite de la memoria a la que accede, lo cual limita las posibilidades de que suceda un desastre si el código se desmanda. Además, al especificar para la base exactamente el punto de inicio con el que queremos trabajar, no es preciso hacer una suma para direccionarla, y el código será algo más rápido.

3 | Otra posibilidad es usar los servicios DPMI para crear un selector a partir de una dirección del modo real. Con este fin la librería de DJGPP nos ofrece la función __dpmi_segment_to_descriptor, la cual es más fácil de usar que la función del caso anterior. El problema es que los selectores creados así tendrán siempre un límite de 64 Kbytes por defecto. Esto, sin embargo, resulta ideal para acceder al segmento-ventana de los 64 Kbytes de vídeo en 0xA000:0000. Crearemos el selector de este modo:



```
short video = __dpmi_segment_to_descriptor(0xa000);
```

Las funciones farpoke() y farpeek() facilitan la escritura y la lectura de valores de 8, 16 y 32 bits en la memoria

Por fin, cuando tengamos el selector podremos usarlo para acceder a la dirección deseada del primer Mega y trastear con ella y la memoria del programa, empleando alguno de los métodos siguientes:

1 | Para insertar o extraer valores de 8, 16 o 32 bits podemos usar las funciones de la familia _farpoke() y farpeek(). Las funciones del primer tipo las usaremos cuando deseemos insertar un valor en una posición de memoria, y las segundas serán útiles cuando queramos leerlo. El formato de, por ejemplo, _farpokeb() es…

```
_farpokeb(unsigned short selector, unsigned long offset, unsigned char valor);
```

Por supuesto la "b" indica que el valor a colocar en la posición direccionada por el selector y el offset es de 8 bits. Con _farpokew() y _farpokel() el formato es idéntico salvo por lo relativo al valor a insertar. En cuanto a _farpeekb(), el formato es…

```
unsigned char valor = _farpeekb(unsigned short selector, unsigned long offset);
```

…cambiando el tamaño del valor a tomar según si empleamos _farpeekw() o _farpeekl().

2 | El inconveniente que tienen las funciones del caso anterior es que en todas ellas tendremos que especificar el selector para cualquier acceso. Una manera más rápida de trabajar con la memoria baja, sobre todo en casos de bucles, será emplear _farsetsel(). Esta función, cuya sintaxis es:

```
void _farsetsel(unsigned short selector);
```

indicará a las funciones de la familia farnspoke() que deberán emplear el selector puesto en _farsetsel(). De esta manera nos ahorraremos el desperdicio de tiempo que supone el indicar el selector en cada acceso. La "ns" del nombre de estas funciones significa que no se indica el selector dentro de la función y éstas usan los mismos sufijos que sus hermanas. O sea: b para 8 bits, w para 16 y l para 32.

3 | Otra posibilidad es usar las funciones dosmemget() y dosmemput() para transferir bloques de datos entre la memoria convencional y la del programa. Estas funciones usan internamente al

DJGPP V.2.1

selector _dos_ds. por lo que no necesitaremos especificarlo. La sintaxis de dosmemget() es:
void dosmemget(int offset, int longitud, void *buffer);
La función transferirá un número de bytes especificado en "longitud" desde la memoria baja en la dirección indicada por el offset hasta la posición apuntada por el puntero buffer en la memoria del programa. Dosmemput() realiza la función inversa y su formato es:
void dosmemput(const void *buffer, int longitud, int offset);

EJEMPLOS, LIBRERÍAS Y UTILIDADES

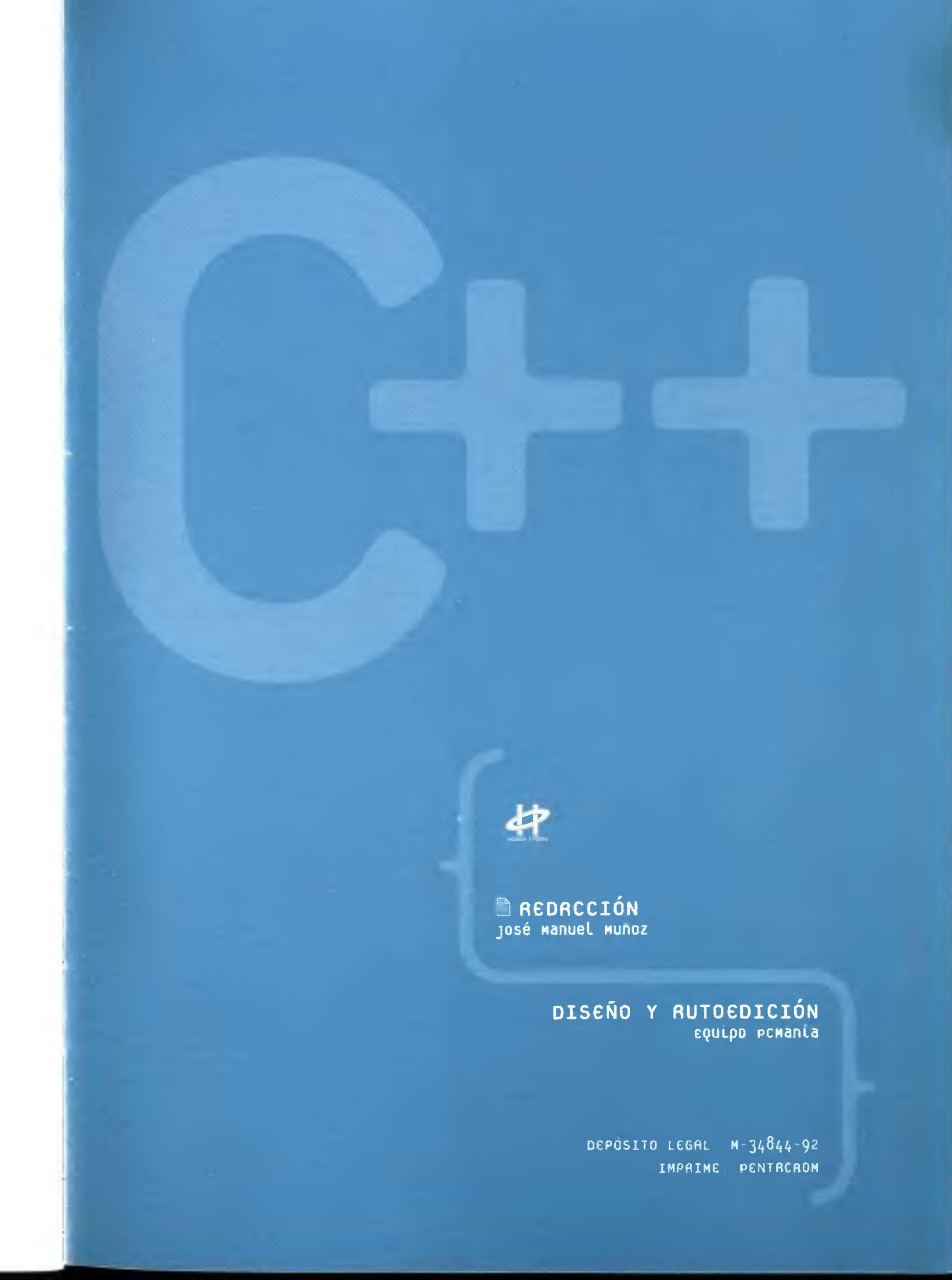
¿Dónde conseguir todo esto? ¿Cuáles son las librerías mas empleadas? Bien, uno de los mejores sitios para encontrar cosas está en ftp://x2ftp.oulu.fi, en el subdirectorio pub/msdos/programming/djgpp2. También os aconsejamos que carguéis el archivo djgppfaq.htm (en el zip faq210b) desde Navigator o Explorer, y exploréis los links indicados en el apartado 22.2 ("Where to find sample DJGPP code or a package ported to DJGPP?"). En cuanto a las librerías, las más importantes parecen ser GRX, Allegro, Xlib y Jlib. Algunas de ellas, como Allegro, son algo más que meras librerías gráficas. Por ejemplo en el caso de Allegro, esta es una librería de funciones escrita en DJGPP usando C y ensamblador y que sirve para crear juegos. Allegro usa el linear frame buffer de VESA, tiene funciones para dibujar polígonos en modo Goraud o aplicar texturas a estos, ofrece scroll por hardware, manipulación de paleta, y soporte para música MIDI, dispone de funciones para gobernar el ratón, el teclado y el joystick y tiene también funciones para manipular matrices y vectores, mover sprites, mostrar archivos FLC, etc. (y en este caso el etc. hace honor a su función). Para quien desee comprobar esto, la página natal de Allegro es http://www.talula.demon.co.uk/allegro.
Este último ejemplo os dará una idea acerca de la calidad de algunas de las librerías y utilidades que pueden encontrarse para DJGPP en la red.

LA NUEVA VERSIÓN

Para concluir haremos un breve resumen de algunas de las nuevas caracteristicas de la versión 2.1 tal como se describen en diversos documentos y también según lo que hemos visto con nuestros propios ojos cuando ello ha sido posible (porque la cantidad de cosas que había que probar es verdaderamente monstruosa):

1) El extensor es ahora más rápido.
2) Los gráficos trabajan ahora incluso bajo Windows (no hemos tenido ocasión de probar esto).
3) Arreglados muchos bugs relacionados con las interrupciones y otras cuestiones del bajo nivel.
4) DJGPP adjunta el programa de libre uso Cwsdpmi que hace innecesario el uso de go32.exe.
5) Aunque esto no forma realmente parte de la nueva versión, hay que añadir que ya están apareciendo las primeras herramientas para crear verdaderas aplicaciones para Windows 95 y NT y otras utilidades muy sabrosas (por ejemplo hemos leído por ahí que ya se pueden crear juegos usando DirectX mediante Allegro).

Y con esto concluimos esta pequeña presentación de la versión 2.1 de DJGPP. Esperamos que os haya resultado útil, y por supuesto que la disfruteis.



REDACCIÓN
José Manuel Muñoz

DISEÑO Y AUTOEDICIÓN
Equipo PCmanía

DEPÓSITO LEGAL M-34844-92
IMPRIME PENTACROM